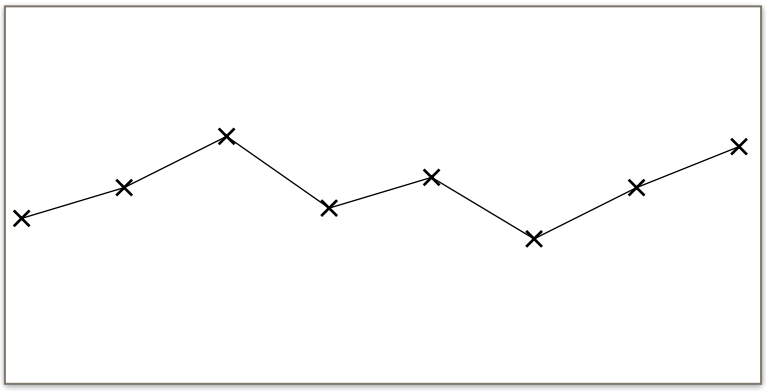
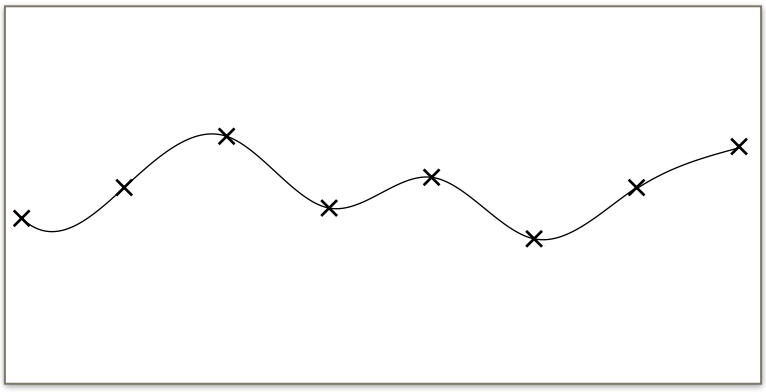
Timo Denk

INTERPOLATION





Mittwoch, 05. November 2014

Inhaltsverzeichnis

Das Titelbild zeigt Beispiele für die lineare (oben) und die kubische (unten) Interpolation bei identischem Datensatz.

Eigenständigkeitserklärung

Hiermit versichere ich, Timo Denk, dass ich die vorliegende schriftliche Ausarbeitung selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen, wie z. B. Internetseiten übernommen habe, habe ich als Zitat mit Angabe der Quelle kenntlich gemacht.

1. Einführung

Als Interpolation bezeichnet man in der numerischen Analysis eine Methode, mit der aus vorgegebenen Datenpunkten eine Funktion berechnet wird, mit der Werte zwischen den Datenpunkten konstruiert werden können.

In Ingenieurs- und Naturwissenschaften tritt häufig die Situation auf, dass eine begrenzte Menge an Datenpunkten vorliegt, beispielsweise bei Messungen. Für diese sind die y-Werte (Messwerte oder Lösungen) für vorgegebene x-Werte (Eingangswerte) verfügbar, benötigt werden aber Lösungen für x-Werte, die zwischen den gemessenen Datenpunkten liegen. Um einen solchen y-Wert zu ermitteln, eignen sich die Methoden der Interpolation oder der Regression. Die vorliegende Arbeit stellt einige Interpolationsmethoden vor.

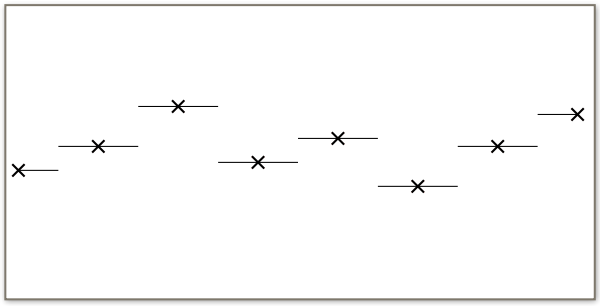
Die Interpolation unterscheidet sich von der Regression dadurch, dass die bei der Interpolation gefundene Funktion durch alle gegebenen Punkte verläuft, während die Regressionsfunktion die beste Annäherung für alle Punkte darstellt und diese dabei nicht notwendigerweise durchläuft. Die mit beiden Techniken verwandte Extrapolation dient dazu, Werte außerhalb des Definitionsbereichs der Punkte zu prognostizieren.

Bei dem Großteil der hier vorgestellten Interpolationsmethoden wird eine Interpolationsfunktion errechnet, die sich aus mehreren Funktionen zusammensetzt. Diese sogenannten Teilfunktionen gelten in den Intervallen zwischen den einzelnen Messpunkten. Die erste Teilfunktion gilt somit beispielsweise für den Bereich zwischen dem x-Wert des ersten Punktes und dem des zweiten Punktes, die zweite Teilfunktion für den Bereich zwischen dem x-Wert des zweiten und des dritten Punktes, usw.

Die erste in dieser Arbeit vorgestellte Interpolationsmethode heißt *Nearest-Neighbor* Interpolation. Ihre Teilfunktionen sind nullten Grades (waagerechte Geraden). Anschließend werden die lineare (Teilfunktionen ersten Grades), quadratische (Teilfunktionen zweiten Grades) und kubische Interpolation (Teilfunktionen dritten Grades) erklärt. Die danach gezeigte Polynominterpolation unterscheidet sich wesentlich von den anderen Methoden, da sich die Interpolationsfunktion nicht aus Teilfunktionen zusammensetzt, sondern ein einzelnes Polynom ist. Abschließend werden verschiedene weitere Interpolationsmethoden kurz erläutert (Gaußprozess, trigonometrische und logarithmische Interpolation).

Bei drei Interpolationsmethoden (der *Nearest-Neighbor*, der linearen und der quadratischen) wird deren Realisierung in JavaScript gezeigt. Als Programmiersprache wurde JavaScript gewählt, da für die Ausführung von JavaScript Code nur ein Webbrowser (wie beispielsweise Google Chrome oder Firefox) benötigt wird, in dessen Konsole man den Programmcode einfügen und ausführen kann. Es wird deshalb kein Compiler oder eine Programmierumgebung benötigt. Die Codebeispiele sind nicht auf Performance optimiert, sondern sollen vor allem dem Verständnis dienen. Die programmiertechnische Lösung der kubischen und der Polynominterpolation sind der der quadratischen sehr ähnlich, weshalb bei diesen beiden Interpolationverfahren auf Codebeispiele verzichtet wurde.

2. Nearest-Neighbor Interpolation



Die *Nearest-Neighbor* (deutsch, sinngemäß: nächstgelegener Punkt) Interpolation ist die einfachste Form der Interpolation. Dabei wird für jeden x-Wert der Wert des nächstgelegenen Datenpunktes als y-Wert gewählt.

Das Verfahren ist rechnerisch äußerst unkompliziert und deshalb für die Interpolation von Punkten geeignet, wenn diese in großer Zahl vorliegen, eher ungenau sind und schnell verarbeitet werden müssen.

Dies ist zum Beispiel bei der Skalierung von Bildern im Browser der Fall, wenn die Bilder einer Website skaliert werden, nachdem sie geladen sind. In diesem Fall werden diese mit der *Nearest-Neighbor* Interpolation interpoliert (sofern eine Interpolation nötig ist). Erst danach werden rechen- und damit zeitaufwändigere Interpolationsverfahren angewandt.

2.1 Allgemein

Für die Punkte

$$P_1(x_1 \mid y_1)$$
$$P_2(x_2 \mid y_2)$$
$$P_3(x_3 \mid y_3)$$
$$\dots$$
$$P_{n-2}(x_{n-2} \mid y_{n-2})$$
$$P_{n-1}(x_{n-1} \mid y_{n-1})$$
$$P_n(x_n \mid y_n)$$

würde gemäß der Regel, dass jedem x-Wert der y-Wert des nächstgelegenen Punktes zugeordnet wird, folgende Interpolationsfunktion gelten:

$$f(x) = \begin{cases} y_1 & \left| x_1 \le x \le \frac{x_1 + x_2}{2} \right. \\ y_2 & \left| \frac{x_1 + x_2}{2} \le x \le \frac{x_2 + x_3}{2} \right. \\ \dots & \\ y_{n-1} & \left| \frac{x_{n-2} + x_{n-1}}{2} \le x \le \frac{x_{n-1} + x_n}{2} \right. \\ y_n & \left| \frac{x_{n-1} + x_n}{2} \le x \le x_n \right. \end{cases}$$

2.2 Code

Das Codebeispiel in der Sprache JavaScript zeigt, wie man die *Nearest-Neighbor* Interpolation mit einem Programmcode realisieren kann.

```
function nearestNeighborInterpolation(xValues, yValues, x) {
      // check if x- and y-values have the same length
      // check if the length is at least two
      if (xValues.length != yValues.length || xValues.length < 2)
            return false;

      // check user values are out of range
      if (xValues[0] >= x)
            return yValues[0];
      if (xValues[xValues.length-1] <= x)
            return yValues[yValues.length-1];


      var shortestDistance, currentDistance;

      for (var i = 0; i < xValues.length; i++) {
            // distance between user x and current (loop) x
            currentDistance = Math.abs(x - xValues[i]);

            if (i == 0 || shortestDistance >= currentDistance)
                  shortestDistance = currentDistance;
            else
                  return yValues[i-1];

      }

      return yValues[xValues.length - 1];
}
```

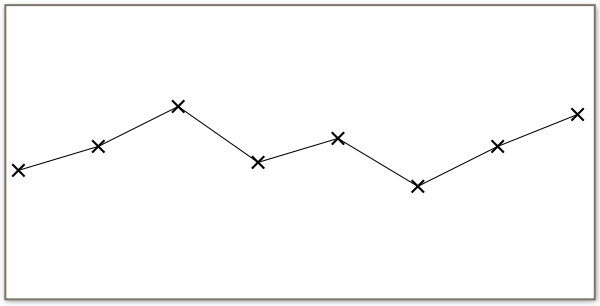
Die Funktion `nearestNeighborInterpolation()` gibt den interpolierten Wert aus den übergebenen Daten-Arrays (`xValues` und `yValues`) an der Stelle `x` zurück. Sie könnte zum Beispiel wie folgt aufgerufen werden:

```
nearestNeighborInterpolation([0, 1, 2, 3], [10, -1, 3, 8], 2.429);
```

Die vorderen 4 Zahlen sind die x-Werte, die darauffolgenden 4 die dazugehörigen y-Werte. Der dritte Parameter gibt den x-Wert an, für den der interpolierte Wert zurückgegeben werden soll.

Der Rückgabewert wäre bei diesem Aufruf `2`.

3. Lineare Interpolation



Bei der linearen Interpolation werden die zu interpolierenden Punkte durch lineare Funktionen miteinander verbunden. So entstehen *n-1* Geradengleichungen, wobei *n* für die Anzahl der Punkte steht. Jede der Gleichungen gilt für die x-Werte, die zwischen zwei bestimmen Punkten liegen.

Verglichen mit anderen Verfahren ist die lineare Interpolation einfach und wenig rechenintensiv. Sie ist damit prädestiniert für die Interpolation von Punkten, die in großer Zahl vorliegen und eine verhältnismäßig geringe Genauigkeit haben.

Eingesetzt wird die lineare Interpolation in vielen Bereichen, beispielsweise in der Astronomie, um die Methode der kleinsten Quadrate (χ^2) auf zwei Lichtkurven von Himmelskörpern anzuwenden. Bevor man die Abstände der beiden Kurven an einem bestimmen Punkt berechnen kann, muss man die beiden Kurven häufig zuerst interpolieren.[1] Auch in Grafikprozessoren wird das Verfahren der linearen Interpolation angewendet, zum Beispiel um Bilder zu skalieren, ohne diese verpixelt darzustellen. Bei neuen Browsern kann der Programmierer sogar selbst einstellen, welche Form der Interpolation verwendet werden soll.[2]

3.1 Allgemein

Um aus den Punkten

$$P_1\left(x_1 \mid y_1\right)$$
$$P_2\left(x_2 \mid y_2\right)$$
$$\dots$$
$$P_{n-1}\left(x_{n-1} \mid y_{n-1}\right)$$
$$P_n\left(x_n \mid y_n\right)$$

die Gleichung der Interpolationsfunktion zu errechnen, müssen die Geradengleichungen für die Geraden zwischen den einzelnen Punkten aufgestellt werden. Zuerst wird für jede Gleichung die Steigung mit

$$m = \frac{\Delta y}{\Delta x}$$

bestimmt und anschließend mit den x- und y-Werten eines Punktes am Rand des zu interpolierenden Intervalls in die Geradengleichung

$$y = mx + c$$

eingesetzt. Der y-Achsenabschnitt (*c*) kann somit errechnet werden und die Gleichung für das jeweiligen Intervall ist bestimmt.

---

[1] http://lcgen.simsso.de

[2] http://entropymine.com/resamplescope/notes/browsers/

Zusammengefasst und umgeformt resultiert daraus die folgende Interpolationsfunktion:

$$f(x) = \begin{cases} y_1 + (y_2 - y_1)\dfrac{x - x_1}{x_2 - x_1} & \left| x_1 \le x \le x_2 \right. \\ \dots & \\ y_{n-1} + (y_n - y_{n-1})\dfrac{x - x_{n-1}}{x_n - x_{n-1}} & \left| x_{n-1} \le x \le x_n \right. \end{cases}$$

3.2 Code

Das Codebeispiel in der Sprache JavaScript zeigt, wie man die lineare Interpolation mit Programmcode realisieren kann.

```
function linearInterpolation(xValues, yValues, x) {
      // check if x- and y-values have the same length
      // check if the length is at least two
      if (xValues.length != yValues.length || xValues.length < 2)
            return false;

      // check if user values are out of range
      if (xValues[0] >= x)
            return yValues[0];
      if (xValues[xValues.length-1] <= x)
            return yValues[yValues.length-1];

      var i = 0;
      // go through all points until the current x value is smaller or
      // equal to the users x-value
      while (xValues[i] <= x) {
            if (xValues[i] == x) // no interpolation necessary
                  return yValues[i];
            i++;
      }

      var x1 = xValues[i-1], x2 = xValues[i];
      var y1 = yValues[i-1], y2 = yValues[i];

      return y1+(y2-y1)*((x-x1)/(x2-x1));
}
```

Die Funktion `linearInterpolation()` gibt den interpolierten Wert aus den übergebenen Daten-Arrays (`xValues` und `yValues`) an der Stelle `x` zurück. Sie könnte zum Beispiel wie folgt aufgerufen werden:

```
linearInterpolation([0, 1, 2, 3], [10, -1, 3, 8], 2.429);
```

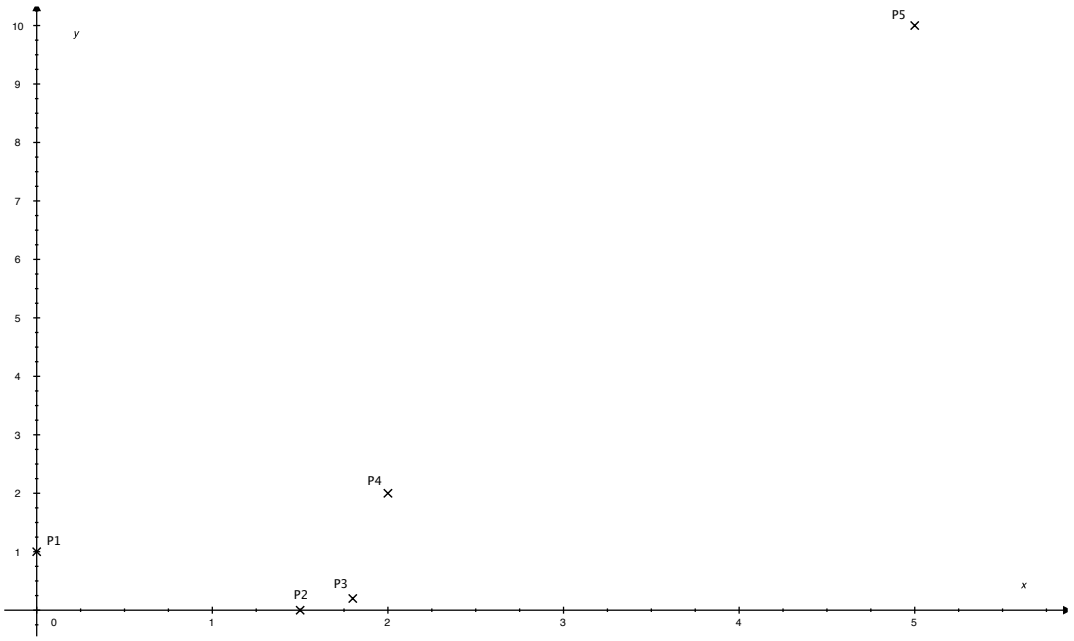
Die vorderen 4 Zahlen sind die x-Werte, die darauffolgenden 4 die dazugehörigen y-Werte. Der zwei Übergabeparameter ist der x-Wert, für den der interpolierte y-Wert bestimmt werden soll.

Der zurückgegebene Wert wäre `5.145`.

3.3 Beispiel

Die nachfolgenden fünf Punkte sollen linear interpoliert werden. Die Vorgehensweise ist dabei ausführlicher dargestellt als in Kap. 3.1, bei der allgemeinen Beschreibung der linearen Interpolation.

$P_1(0\,|\,1)$
$P_2(1{,}5\,|\,0)$
$P_3(1{,}8\,|\,0{,}2)$
$P_4(2\,|\,2)$
$P_5(5\,|\,10)$



Zuerst werden die vier Funktionen für die Geraden zwischen den Punkten aufgestellt. Diese enthalten bisher nur Variablen. Die erste Funktion liegt zwischen dem ersten und dem zweiten Punkt, die zweite zwischen dem zweiten und dritten Punkt, usw.

$$f_1(x) = m_1 x + c_1$$
$$f_2(x) = m_2 x + c_2$$
$$f_3(x) = m_3 x + c_3$$
$$f_4(x) = m_4 x + c_4$$

Anschließend werden deren Steigungen *m* berechnet. Dazu wird *Δy* durch *Δx* geteilt. *Δy* berechnet sich aus dem y-Wert des rechten Punkts minus dem y-Wert des linken Punkts. Für *Δx* gilt dasselbe mit den x-Werten der beiden Punkte. Für m_1 wird folglich der y-Wert des zweiten Punktes minus den y-Wert des ersten Punktes verwendet. Dieses Ergebnis wird durch den x-Wert des zweiten minus den x-Wert des ersten Punktes geteilt. Die so errechnete Steigung gilt für die Gerade im Bereich zwischen den beiden Punkten. Für alle Geradensteigungen des Beispiels werden die Werte wie folgt ausgerechnet:

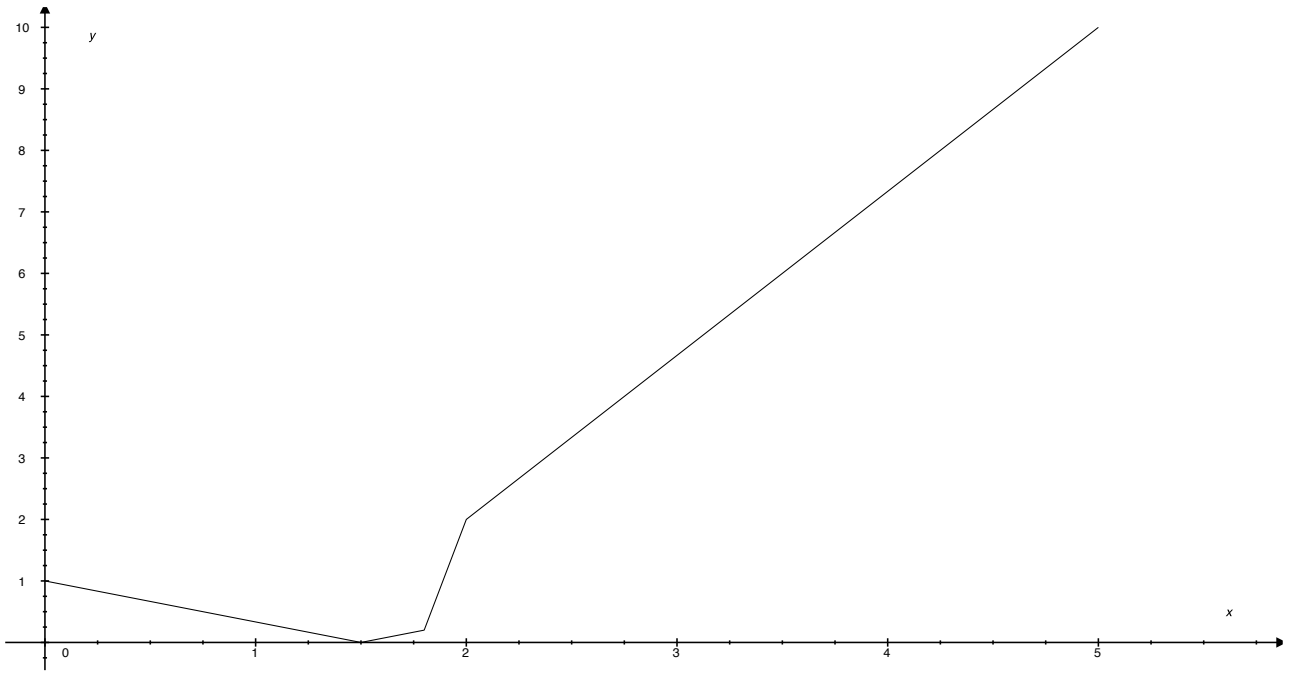
$$m_1 = \frac{y_2 - y_1}{x_2 - x_1} = \frac{0-1}{1{,}5-0} = -\frac{2}{3}$$
$$m_2 = \frac{y_3 - y_2}{x_3 - x_2} = \frac{0{,}2-0}{1{,}8-1{,}5} = \frac{2}{3}$$
$$m_3 = \frac{y_4 - y_3}{x_4 - x_3} = \frac{2-0{,}2}{2-1{,}8} = 9$$
$$m_4 = \frac{y_5 - y_4}{x_5 - x_4} = \frac{10-2}{5-2} = \frac{8}{3}$$

In jeder der einzelnen Geradengleichungen liegt jetzt nur noch die Unbekannte *c* vor. Man kann *c* berechnen, indem man *m* in die jeweilige Geradengleichung einsetzt, sowie den x- und y-Wert von einem der beiden Punkte, durch den die Gerade verläuft. Bei der ersten Gerade könnte man beispielsweise den x- und den y-Wert des ersten Punktes einsetzten, bei der zweiten die Werte des zweiten Punktes, usw.:

$$f_1(0)=1 \Rightarrow 1=-\frac{2}{3}\cdot 0+c_1 \Rightarrow c_1=1$$
$$f_2(1{,}5)=0 \Rightarrow 0=\frac{2}{3}\cdot 1{,}5+c_2 \Rightarrow c_2=-1$$
$$f_3(1{,}8)=0{,}2 \Rightarrow 0{,}2=9\cdot 1{,}8+c_3 \Rightarrow c_3=-16$$
$$f_4(2)=2 \Rightarrow 2=\frac{8}{3}\cdot 2+c_4 \Rightarrow c_4=-\frac{10}{3}$$

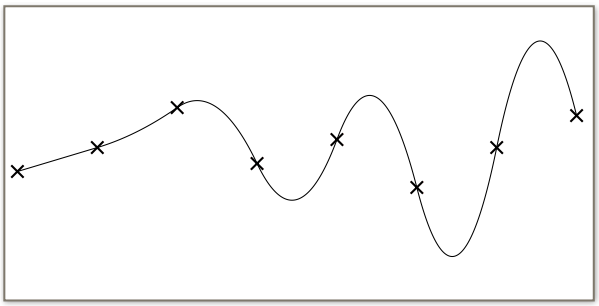
Alle Unbekannten (*m* und *c*) in den vier Geradengleichungen sind jetzt bestimmt, und die Interpolationsfunktion kann aus den vier Gleichungen zusammengesetzt werden. Die jeweiligen Geraden gelten immer nur in den Bereichen zwischen den beiden Punkten, die sie durchlaufen, weshalb hinter den Teilfunktionen der Funktion *f(x)* die Gültigkeitsbereiche stehen.

$$f(x)=\begin{cases} -\frac{2}{3}x+1 & |0\le x\le 1{,}5 \\ \frac{2}{3}x-1 & |1{,}5\le x\le 1{,}8 \\ 9x-16 & |1{,}8\le x\le 2 \\ \frac{8}{3}x-\frac{10}{3} & |2\le x\le 5 \end{cases}$$



Die Abbildung zeigt den Graph der Interpolationsfunktion.

4. Quadratische Interpolation



Bei der quadratischen Interpolation werden die zu interpolierenden Punkte durch quadratische Funktionen miteinander verbunden. Eine wesentliche Bedingung ist, dass die quadratischen Funktionen in den Punkten dieselbe Steigung wie ihre Nachbarfunktionen haben. So entstehen *n-1* quadratische Gleichungen, wobei *n* für die Anzahl der Punkte steht. Jede der Gleichungen gilt für die x-Werte, die zwischen zwei bestimmen Punkten liegen.

Die quadratische Interpolation ist erheblich komplizierter als die lineare und trotzdem nicht unbedingt besser als diese. Geeignet ist die quadratische Interpolation für Daten, die keine großen Ausschläge in y-Richtung haben; einen eher flachen Verlauf haben. Die Abb. oben auf dieser Seite zeigt sehr gut, dass die Ausschläge der quadratischen Teilfunktionen sonst sehr stark sein können (insbesondere zwischen den letzten fünf Punkten), was bei den meisten Anwendungen ein unerwünschter Effekt ist.[3] Der Grund dafür ist, dass quadratische Funktionen keine Wendepunkte haben.

4.1 Allgemein

Um die Punkte

$$P_1(x_1 \mid y_1)$$
$$P_2(x_2 \mid y_2)$$
$$P_3(x_3 \mid y_3)$$
$$P_4(x_4 \mid y_4)$$
$$\dots$$
$$P_{n-1}(x_{n-1} \mid y_{n-1})$$
$$P_n(x_n \mid y_n)$$

quadratisch zu interpolieren, stellt man die quadratischen Teilfunktionen

$$f_1(x) = a_1x^2 + b_1x + c_1$$
$$f_2(x) = a_2x^2 + b_2x + c_2$$
$$f_3(x) = a_3x^2 + b_3x + c_3$$
$$\dots$$
$$f_{n-1}(x) = a_{n-1}x^2 + b_{n-1}x + c_{n-1}$$

[3] http://www.korf.co.uk/spline.pdf

auf. Sie sind zweiten Grades und haben je drei Unbekannte. Jede der Teilfunktionen liegt zwischen zwei Punkten, die sie durchläuft, weshalb man auf die Gleichungen

$$f_1(x_1) = y_1$$
$$f_1(x_2) = y_2$$
$$f_2(x_2) = y_2$$
$$f_2(x_3) = y_3$$
$$f_3(x_3) = y_3$$
$$f_3(x_4) = y_4$$
$$\dots$$
$$f_{n-1}(x_{n-1}) = y_{n-1}$$
$$f_{n-1}(x_n) = y_n$$

schließen kann. Zusätzlich ist eine Bedingung, dass die quadratischen Funktionen in den einzelnen Punkten die gleiche Steigung haben wie die anliegende quadratische Funktion. Im Punkt P_2 haben also beispielsweise die Funktionen f_1 und f_2 die gleiche Steigung. Somit lassen sich die Gleichungen

$$\frac{d}{dx}\left(f_1(x)\right)\Big|_{x=x_2} = \frac{d}{dx}\left(f_2(x)\right)\Big|_{x=x_2}$$
$$\frac{d}{dx}\left(f_2(x)\right)\Big|_{x=x_3} = \frac{d}{dx}\left(f_3(x)\right)\Big|_{x=x_3}$$
$$\dots$$
$$\frac{d}{dx}\left(f_{n-1}(x)\right)\Big|_{x=x_{n-1}} = \frac{d}{dx}\left(f_{n-2}(x)\right)\Big|_{x=x_{n-1}}$$

aufstellen. Nach dem Umformen zu

$$2a_1x_2 + b_1 = 2a_2x_2 + b_2$$
$$2a_2x_3 + b_2 = 2a_3x_3 + b_3$$
$$\dots$$
$$2a_{n-2}x_{n-1} + b_{n-2} = 2a_{n-1}x_{n-1} + b_{n-1}$$

liegt insgesamt nur noch eine Gleichung zu wenig vor. Eine Möglichkeit, diese zu gewinnen, ist es, $a_1 = 0$ zu setzen, was bewirkt, dass die erste Funktion (f_1) linear ist. Es gibt auch andere Methoden zur Gewinnung der zusätzlich benötigten Gleichung, die in Kap. 5.1 erklärt werden. Es ist aber zu beachten, dass sich die dort erklärten Methoden auf die kubische Interpolation beziehen und weshalb eine Adaption erforderlich ist. Unabhängig davon, welche Technik man verwendet, kann man das Gleichungssystem danach eindeutig lösen. Nach dem Einsetzen der Werte in die Teilfunktionen erhält man die Interpolationsfunktion:

$$f(x) = \begin{cases} a_1x^2 + b_1x + c_1 & \left| x_1 \le x \le x_2 \right. \\ a_2x^2 + b_2x + c_2 & \left| x_2 \le x \le x_3 \right. \\ a_3x^2 + b_3x + c_3 & \left| x_3 \le x \le x_4 \right. \\ \dots & \\ a_{n-1}x^2 + b_{n-1}x + c_{n-1} & \left| x_{n-1} \le x \le x_n \right. \end{cases}$$

4.2 Code

Das Codebeispiel zeigt, wie man eine quadratische Interpolation in JavaScript realisieren kann. Der Code, der zur programmiertechnischen Lösung der kubischen Interpolation erforderlich wäre, unterscheidet sich nicht wesentlich vom hier aufgelisteten; ähnlich verhält es sich mit der Polynominterpolation. Um so größer ist der Unterschied zur linearen und zur Nearest-Neighbor Interpolation, da bei diesen beiden Verfahren keine Lösung von komplexen Gleichungssystemen erforderlich ist. Die dazu geschriebene `solveMatrix()` Funktion tut ebendas (ein als Matrix ausgedrücktes Gleichungssystem lösen) und ist nachfolgend aufgelistet. Zur Lösung der Matrix wird innerhalb der Funktion das Gaußsche Eliminationsverfahren verwendet.

```
function solveMatrix(mat) {
    var len = mat.length;
    for (var i = 0; i < len; i++) // column
        for (var j = i+1; j < len; j++) {// row
            if (mat[i][i] == 0) { // check if cell is zero
                var k = i;

                // search for an element where this cell is not zero
                while (mat[k][i] == 0) k++;

                // swap rows
                var tmp = mat[k].slice();
                mat[k] = mat[i].slice();
                mat[i] = tmp.slice();
            }

            var fac = -mat[j][i]/mat[i][i];
            for(var k = i; k < len+1; k++) // elements in a row
                mat[j][k] += fac *mat[i][k];
        }

    var solution = [];
    for (var i = len-1; i >= 0; i--) { // column
        solution.unshift(mat[i][len]/mat[i][i]);
        for (var k = i-1; k >= 0; k--)
            mat[k][len] -= mat[k][i] * solution[0];
    }
    return solution;
}
```

Die eigentliche Funktion für die quadratische Interpolation heißt `quadraticInterpolation()`. Sie errechnet aus einem Daten-Array, das aus Objekten mit den Attributen `x` und `y` besteht, den quadratisch interpolierten Wert an der Stelle `x` und loggt die Koeffizienten der einzelnen Teilfunktionen in der Konsole. Dazu wird zuerst eine Matrix aufgestellt, die alle Gleichungen enthält. Die Hilfsfunktion löst diese Matrix, und anschließend erfolgt die Ausgabe des berechneten Funktionswertes *y*.

```
function quadraticInterpolation(data, x) {
    // check if user values are out of range
    if (x < data[0].x || x > data[data.length-1].x)
        return undefined;

    // initialize matrix
    var numberOfUnknowns = 3*(data.length-1);
    var numberOfFunctions = data.length-1;
    var numberOfDerivatives = data.length-2;
```

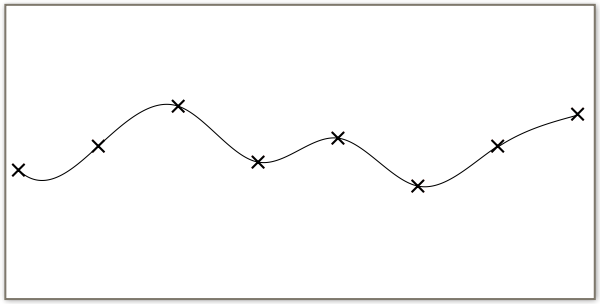
```
    var matrix = new Array(numberOfUnknowns);
    for (var i = 0; i < matrix.length; i++) {
        matrix[i] = new Array(numberOfUnknowns+1);
        for (var j = 0; j < matrix[i].length; j++)
            matrix[i][j] = 0;
    }

    // through points equations
    for (var i = 0, j = 0; j < data.length-1; i += 2, j++) {
        matrix[i][j*3] = Math.pow(data[j].x, 2);
        matrix[i][j*3+1] = data[j].x;
        matrix[i][j*3+2] = 1;
        matrix[i][numberOfUnknowns] = data[j].y;
        matrix[i+1][j*3] = Math.pow(data[j+1].x, 2);
        matrix[i+1][j*3+1] = data[j+1].x;
        matrix[i+1][j*3+2] = 1;
        matrix[i+1][numberOfUnknowns] = data[j+1].y;
    }

    // derivative equations
    for (var i=(data.length-1)*2, j=0; i<numberOfUnknowns-1; i++, j++) {
        matrix[i][(j*3)] = 2*data[j+1].x;
        matrix[i][(j*3)+1] = 1;
        matrix[i][(j*3)+3] = -2*data[j+1].x;
        matrix[i][(j*3)+4] = -1;
    }

    // additional information / boundary condition (required)
    matrix[numberOfUnknowns-1][0] = 1; // make first function linear

    var coefficients = solveMatrix(matrix);
    console.log(coefficients);

    var i = 0;
    while (data[i].x <= x) {
        if (data[i].x == x)
            return data[i].y;
        i++;
    }
    i--;

    return coefficients[i*3]*Math.pow(x,2)+coefficients[i*3+1]*x
+coefficients[i*3+2];
}
```

Ein Funktionsaufruf könnte zum Beispiel so aussehen:

```
quadraticInterpolation(
      [
            { x: 0, y: 1 },
            { x: 1, y: .2 },
            { x: 2, y: .5 },
            { x: 3, y: .5 }
      ],
      2.001
);
```

Der zurückgegebene Wert wäre dann `0.5013985999999999`.

5. Kubische Interpolation



Bei der kubischen Interpolation werden die zu interpolierenden Punkte durch kubische Funktionen miteinander verbunden. Die kubischen Teilfunktionen sind in den Punkten bezüglich der ersten und zweiten Ableitung identisch zur nebenliegenden Teilfunktion. So ergeben sich *n-1* kubische Gleichungen, wobei *n* wieder für die Anzahl der Punkte steht. Jede der Gleichungen gilt für die x-Werte, die zwischen zwei bestimmen Punkten liegen.

Die kubische Interpolation ist relativ kompliziert, da die zu lösenden Gleichungssysteme im Verhältnis zur Anzahl der Punkte sehr viele Gleichungen enthalten. Das optische Ergebnis ist dafür von sehr guter Qualität, wie auf der Abb. oben auf dieser Seite zu sehen ist. Die Interpolationsfunktion durchläuft die Punkte mit viel geringeren Ausschlägen als die quadratische Interpolationsfunktion (Kap. 4), weil die kubischen Teilfunktionen Wendepunkte enthalten können. Die kubische Interpolation eignet sich daher beispielsweise für Anwendungsbereiche, in denen die interpolierte Funktion zur grafischen Darstellung, zum Beispiel bei Temperaturkurven der Wettervorhersage, dient.

Wenn man die Ausschläge noch weiter minimieren möchte, kann man den Grad der einzelnen Teilfunktionen weiter erhöhen. Man muss dann aber nicht mehr nur die ersten beiden Ableitungen in den inneren Punkten gleichsetzen, um die benötigte Anzahl an Gleichungen zu erhalten, sondern auch Ableitungen höherer Ordnung. Die Komplexität nimmt quadratisch mit dem Grad der Teilfunktionen zu.

5.1 Allgemein

Um die Punkte

$$P_1\left(x_1 \mid y_1\right)$$
$$P_2\left(x_2 \mid y_2\right)$$
$$P_3\left(x_3 \mid y_3\right)$$
$$P_4\left(x_4 \mid y_4\right)$$
$$\dots$$
$$P_{n-1}\left(x_{n-1} \mid y_{n-1}\right)$$
$$P_n\left(x_n \mid y_n\right)$$

kubisch zu interpolieren, stellt man die kubischen Funktionen

$$f_1(x) = a_1x^3 + b_1x^2 + c_1x + d_1$$
$$f_2(x) = a_2x^3 + b_2x^2 + c_2x + d_2$$
$$f_3(x) = a_3x^3 + b_3x^2 + c_3x + d_3$$
$$\dots$$
$$f_{n-1}(x) = a_{n-1}x^3 + b_{n-1}x^2 + c_{n-1}x + d_{n-1}$$

für die Intervalle zwischen den Punkten auf.

Da bekannt ist, dass jede der kubischen Funktionen (von f_1 bis f_{n-1}) zwei bestimmte Punkte hat, durch die sie verläuft, kann man diese beiden Punkte in die jeweilige Gleichung einsetzen und erhält daraus *2(n-1)* Gleichungen:

$$f_1(x_1) = y_1$$
$$f_1(x_2) = y_2$$
$$f_2(x_2) = y_2$$
$$f_2(x_3) = y_3$$
$$f_3(x_3) = y_3$$
$$f_3(x_4) = y_4$$
$$\dots$$
$$f_{n-1}(x_{n-1}) = y_{n-1}$$
$$f_{n-1}(x_n) = y_n$$

Im Gegensatz zur quadratischen Interpolation (Kap. 4) werden nun die Funktionen in den Punkten nicht mehr nur in ihrer ersten, sondern zusätzlich auch in ihrer zweiten Ableitung gleichgesetzt. Man erhält dadurch für die erste Ableitung folgende Gleichungen:

$$\frac{d}{dx}\left(f_1(x)\right)\Big|_{x=x_2} = \frac{d}{dx}\left(f_2(x)\right)\Big|_{x=x_2}$$
$$\frac{d}{dx}\left(f_2(x)\right)\Big|_{x=x_3} = \frac{d}{dx}\left(f_3(x)\right)\Big|_{x=x_3}$$
$$\dots$$
$$\frac{d}{dx}\left(f_{n-1}(x)\right)\Big|_{x=x_{n-1}} = \frac{d}{dx}\left(f_{n-2}(x)\right)\Big|_{x=x_{n-1}}$$

Für die zweite Ableitung lassen sich die Gleichungen

$$\frac{d}{dx}\left(\frac{d}{dx}\left(f_1(x)\right)\right)\Big|_{x=x_2} = \frac{d}{dx}\left(\frac{d}{dx}\left(f_2(x)\right)\right)\Big|_{x=x_2}$$
$$\frac{d}{dx}\left(\frac{d}{dx}\left(f_2(x)\right)\right)\Big|_{x=x_3} = \frac{d}{dx}\left(\frac{d}{dx}\left(f_3(x)\right)\right)\Big|_{x=x_3}$$
$$\dots$$
$$\frac{d}{dx}\left(\frac{d}{dx}\left(f_{n-1}(x)\right)\right)\Big|_{x=x_{n-1}} = \frac{d}{dx}\left(\frac{d}{dx}\left(f_{n-2}(x)\right)\right)\Big|_{x=x_{n-1}}$$

aufstellen. Die Ableitungsgleichungen gelten immer nur in den Punkten, in denen sich die beiden Funktionen schneiden. Deshalb befindet sich hinter jedem Term eine Angabe, die aussagt, für welchen x-Wert dieser gültig ist. Nach dem Umformen zu

$$3a_1x_2^{\,2} + 2b_1x_2 + c_1 = 3a_2x_2^{\,2} + 2b_2x_2 + c_2$$
$$3a_2x_3^{\,2} + 2b_2x_3 + c_2 = 3a_3x_3^{\,2} + 2b_3x_3 + c_3$$
$$\dots$$
$$3a_{n-2}x_{n-1}^{\,2} + 2b_{n-2}x_{n-1} + c_{n-2} = 3a_{n-1}x_{n-1}^{\,2} + 2b_{n-1}x_{n-1} + c_{n-1}$$

für die erste Ableitung und

$$6a_1x_2 + 2b_1 = 6a_2x_2 + 2b_2$$
$$6a_2x_3^2 + 2b_2 = 6a_3x_3^2 + 2b_3$$
$$\dots$$
$$6a_{n-2}x_{n-1}^2 + 2b_{n-2} = 6a_{n-1}x_{n-1}^2 + 2b_{n-1}$$

für die zweite Ableitung werden nun noch zwei Gleichungen benötigt, um die Interpolationsfunktion eindeutig bestimmen zu können. Es gibt verschiedene Möglichkeiten, diese sogenannten Randbedingungen zu erhalten:[4]

- Die Koeffizienten a_1 und a_{n-1} gleich null setzen, die erste und die letzte Teilfunktion also als quadratisch definieren:

  $$a_1 = 0$$
  $$a_{n-1} = 0$$

- *Natural Spline*: Die zweite Ableitung der ersten und letzten Funktion wird in den Randpunkten gleich null gesetzt. Grafisch bewirkt die Anwendung des *Natural Spline*, dass die Steigungsänderung in den Randpunkten null ist; die Kurve gerade ausläuft.

  $$\frac{d}{dx}\left(\frac{d}{dx}\left(f_1(x)\right)\right) = 0 \quad \Big| x = x_1$$
  $$\frac{d}{dx}\left(\frac{d}{dx}\left(f_{n-1}(x)\right)\right) = 0 \quad \Big| x = x_n$$

- *Periodic Spline*: Die erste Ableitung von Anfangs- und Endpunkt wird, ebenso wie deren zweite Ableitung, gleichgesetzt. Die so entstehende Interpolationsfunktion ist periodisch; sie kann hintereinander gereiht werden, ohne dass Knicke zu sehen sind:

  $$\frac{d}{dx}\left(f_1(x)\right)\Big|_{x=x_1} = \frac{d}{dx}\left(f_{n-1}(x)\right)\Big|_{x=x_n}$$
  $$\frac{d}{dx}\left(\frac{d}{dx}\left(f_1(x)\right)\right)\Big|_{x=x_1} = \frac{d}{dx}\left(\frac{d}{dx}\left(f_{n-1}(x)\right)\right)\Big|_{x=x_n}$$

- *Not-a-Knot Spline*: Die dritte Ableitung von f_1 wird mit der dritten Ableitung von f_2 im zweiten Punkt gleichgesetzt. Ebenso wird mit dem vorletzten Punkt verfahren.

  $$\frac{d}{dx}\left(\frac{d}{dx}\left(\frac{d}{dx}\left(f_1(x)\right)\right)\right)\Big|_{x=x_2} = \frac{d}{dx}\left(\frac{d}{dx}\left(\frac{d}{dx}\left(f_2(x)\right)\right)\right)\Big|_{x=x_2}$$
  $$\frac{d}{dx}\left(\frac{d}{dx}\left(\frac{d}{dx}\left(f_{n-2}(x)\right)\right)\right)\Big|_{x=x_{n-1}} = \frac{d}{dx}\left(\frac{d}{dx}\left(\frac{d}{dx}\left(f_{n-1}(x)\right)\right)\right)\Big|_{x=x_{n-1}}$$

Die jetzt vorliegenden Gleichungen lassen sich in einer Matrix darstellen. Die Zeilenanzahl der Matrix berechnet sich aus $4n$ (n entspricht der Anzahl an Datenpunkten), die Anzahl an Spalten ist

[4] http://www.maths.lth.se/na/courses/FMN081/FMN081-06/lecture11.pdf

um eins höher (*4n+1*). Die Zellenanzahl berechnet sich aus $16n^2+4n$; die Größe der Matrix nimmt quadratisch mit der Anzahl an Punkten zu.

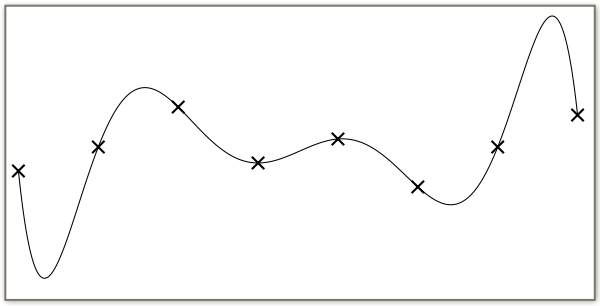
a_1	b_1	c_1	d_1	a_2	b_2	c_2	d_2	a_3	b_3	c_3	d_3	…	a_{n-2}	b_{n-2}	c_{n-2}	d_{n-2}	a_{n-1}	b_{n-1}	c_{n-1}	d_{n-1}	
x_1^3	x_1^2	x_1	1	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	y_1
x_2^3	x_2^2	x_2	1	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	y_2
0	0	0	0	x_2^3	x_2^2	x_2	1	0	0	0	0	⋯	0	0	0	0	0	0	0	0	y_2
0	0	0	0	x_3^3	x_3^2	x_3	1	0	0	0	0	⋯	0	0	0	0	0	0	0	0	y_3
0	0	0	0	0	0	0	0	x_3^3	x_3^2	x_3	1	⋯	0	0	0	0	0	0	0	0	y_3
0	0	0	0	0	0	0	0	x_4^3	x_4^2	x_4	1	⋯	0	0	0	0	0	0	0	0	y_4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	0	0	…	x_{n-2}^3	x_{n-2}^2	x_{n-2}	1	0	0	0	0	y_{n-2}
0	0	0	0	0	0	0	0	0	0	0	0	…	x_{n-1}^3	x_{n-1}^2	x_{n-1}	1	0	0	0	0	y_{n-1}
0	0	0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0	x_{n-1}^3	x_{n-1}^2	x_{n-1}	1	y_{n-1}
0	0	0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0	x_n^3	x_n^2	x_n	1	y_n
$3x_2^2$	$2x_2$	1	0	$-3x_2^2$	$-2x_2$	−1	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0
0	0	0	0	$3x_3^2$	$2x_3$	1	0	$-3x_3^2$	$-2x_3$	−1	0	…	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	0	0	…	$3x_{n-1}^2$	$2x_{n-1}$	1	0	$-3x_{n-1}^2$	$-2x_{n-1}$	−1	0	0
$6x_2$	2	0	0	$-6x_2$	−2	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0
0	0	0	0	$6x_3$	2	0	0	$-6x_3$	−2	0	0	…	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	0	0	…	$6x_{n-1}$	2	0	0	$-6x_{n-1}$	−2	0	0	0
$6x_1$	2	0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0	$6x_n$	2	0	0	0

Die erste Zeile listet die Koeffizienten auf, für die die jeweilige Spalte gilt; sie sind in der endgültigen Matrix wegzulassen. Die zweite Zeilengruppe (Zeile 2 bis 12) repräsentiert die Gleichungen, die aussagen, dass die Teilfunktionen durch die einzelnen Punkte gehen. Die nachfolgenden vier Zeilen (13 bis 16) stehen für die erste Ableitung, die darauffolgenden vier (17 bis 20) für die zweite Ableitung. Die letzten beiden Zeilen enthalten die zusätzlich benötigen Informationen, deren Gewinnung vorher erläutert wurde. Hier wurde ein *Natural Spline* gewählt, diese Zeilen können also variieren.

Nachdem man das Gleichungssystem gelöst hat, kann man die Interpolationsfunktion zusammensetzen:

$$f(x)=\begin{cases} a_1x^3+b_1x^2+c_1x+d_1 & \left| x_1 \le x \le x_2 \right. \\ a_2x^3+b_2x^2+c_2x+d_2 & \left| x_2 \le x \le x_3 \right. \\ a_3x^3+b_3x^2+c_3x+d_3 & \left| x_3 \le x \le x_4 \right. \\ \dots & \\ a_{n-1}x^3+b_{n-1}x^2+c_{n-1}x+d_{n-1} & \left| x_{n-1} \le x \le x_n \right. \end{cases}$$

6. Polynominterpolation



Bei der Polynominterpolation wird ein Polynom gesucht, das durch alle zu interpolierenden Punkte verläuft. Das Polynom ist *n-1*-ten Grades, wobei *n* für die Anzahl der Punkte steht.

Die Polynominterpolation ist aufgrund ihrer starken Ausschläge für viele Anwendungsgebiete ungeeignet. Die rechte Abbildung und das Beispiel in Kap. 6.2 zeigen, wie stark diese Ausschläge sein können. In der digitalen Signalverarbeitung wird die Polynominterpolation zum Beispiel zur Abtastratenkonvertierung (engl.: *Resampling*) verwendet.[5]

6.1 Allgemein

Um das Interpolationspolynom für die Punkte

$P_1(x_1 \mid y_1)$
$P_2(x_2 \mid y_2)$
…
$P_{n-1}(x_{n-1} \mid y_{n-1})$
$P_n(x_n \mid y_n)$

zu bestimmen, werden diese in die Gleichung des Polynoms

$$f(x) = \sum_{i=0}^{n-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$$

eingesetzt. Die höchste Potenz (der Grad des Polynoms) ist dabei um eins niedriger als die Anzahl der Punkte (*n-1*).

$$f(x_1) = y_1 \Rightarrow y_1 = a_0 + a_1 x_1 + a_2 {x_1}^2 + \ldots + a_{n-1} {x_1}^{n-1}$$
$$f(x_2) = y_2 \Rightarrow y_2 = a_0 + a_1 x_2 + a_2 {x_2}^2 + \ldots + a_{n-1} {x_2}^{n-1}$$
…
$$f(x_{n-1}) = y_{n-1} \Rightarrow y_{n-1} = a_0 + a_1 x_{n-1} + a_2 {x_{n-1}}^2 + \ldots + a_{n-1} {x_{n-1}}^{n-1}$$
$$f(x_n) = y_n \Rightarrow y_n = a_0 + a_1 x_n + a_2 {x_n}^2 + \ldots + a_{n-1} {x_n}^{n-1}$$

Das Gleichungssystem kann nun eindeutig gelöst werden, da die Anzahl der Gleichungen der Anzahl an Unbekannten entspricht.

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \quad \left| x_1 \le x \le x_n \right.$$

[5] http://www.cs.tut.fi/kurssit/TLT-5806/Interpol.pdf

6.2 Beispiel

Aus den nachfolgenden fünf Punkten soll das Interpolationspolynom berechnet werden.

$P_1(0\,|\,1)$
$P_2(1{,}5\,|\,0)$
$P_3(1{,}8\,|\,0{,}2)$
$P_4(2\,|\,2)$
$P_5(5\,|\,10)$

Das Polynom ist vierten Grades, da fünf Punkte vorliegen.

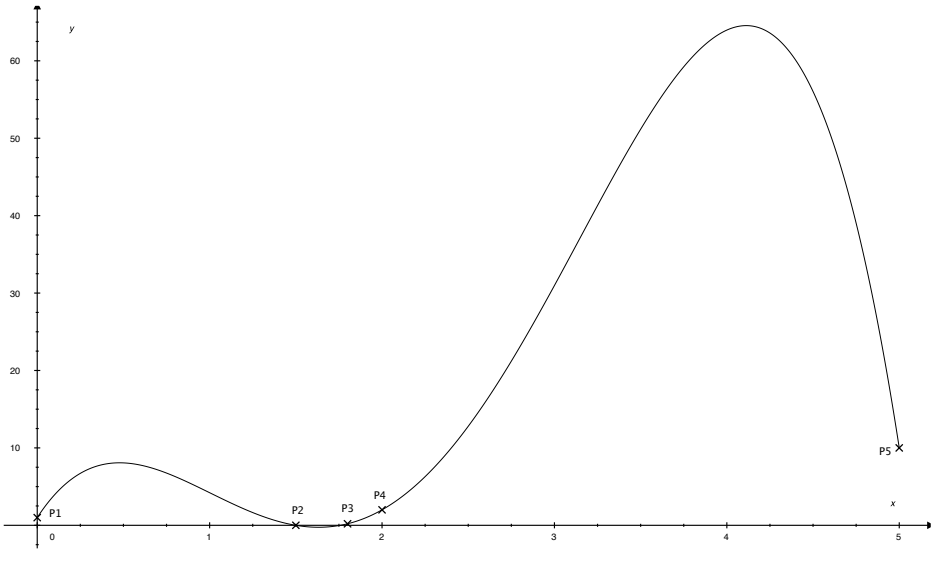
$$f(x) = ax^4 + bx^3 + cx^2 + dx + e$$

Jeder der Punkte wird nun in die Funktion des Polynoms eingesetzt, woraus sich fünf Gleichungen ergeben.

$$f(0) = 1 \Rightarrow 1 = e$$
$$f(1{,}5) = 0 \Rightarrow 0 = \frac{81}{16}a + \frac{27}{8}b + \frac{9}{4}c + \frac{3}{2}d + e$$
$$f(1{,}8) = 0{,}2 \Rightarrow 0{,}2 = \frac{6561}{625}a + \frac{729}{125}b + \frac{81}{25}c + \frac{9}{5}d + e$$
$$f(2) = 2 \Rightarrow 2 = 16a + 8b + 4c + 2d + e$$
$$f(5) = 10 \Rightarrow 10 = 625a + 125b + 25c + 5d + e$$

Das Gleichungssystem lässt sich jetzt eindeutig lösen und man erhält folgende Werte für die Koeffizienten.

$$a = -\frac{4019}{1512}$$
$$b = \frac{333407}{15120}$$
$$c = -\frac{759887}{15120}$$
$$d = \frac{85871}{2520}$$
$$e = 1$$



Setzt man diese Werte in die Funktion des Interpolationspolynoms ein, führt das zur Ergebnisfunktion, deren Graph im Schaubild oben zu sehen ist.

$$f(x) = -\frac{4019}{1512}x^4 + \frac{333407}{15120}x^3 - \frac{759887}{15120}x^2 + \frac{85871}{2520}x + 1 \qquad |0 \le x \le 5$$

7. Weitere Interpolationsverfahren

Neben den vorgestellten Interpolationsverfahren existieren noch viele weitere, von denen drei in diesem Kapitel kurz vorgestellt werden.

7.1 Gaußprozess

Der sogenannte Gaußprozess ist für viele Anwendungsbereiche geeignet und wird unter anderem zur Interpolation, zur Extrapolation oder zur Glättung von Messpunkten eingesetzt.[6] Die Methode wurde nicht von dem Mathematiker Carl-Friedrich Gauß erfunden, aber nach ihm benannt, weil in ihr die Gaußverteilung von mathematischen Funktionen eine wesentliche Rolle spielt.

Die Besonderheit der Methode des Gaußprozesses liegt darin, dass sie nahezu vollständig auf klassischer Wahrscheinlichkeitsrechnung beruht und zumeist besser als andere Methoden die wahrscheinlichsten Werte und auch Unsicherheiten vorhersagen kann. Sie verwendet im wesentlichen Konzepte aus der linearen Algebra und der Gaußschen Fehlerrechnung. Dadurch behält ihr mathematischer Vorgang eine hohe Transparenz und Verständlichkeit.

7.2 Trigonometrische Interpolation

Die trigonometrische Interpolation ist für die Interpolation von Daten, denen periodischen Funktionen zugrunde liegen, geeignet. Es werden dazu trigonometrische Funktionen (Sinus und Cosinus) mit gegebenen Periodenlängen kombiniert, um ein sogenanntes trigonometrisches Polynom zu erhalten, das durch alle Punkte verläuft.

Die allgemeine Form eines trigonometrischen Polynoms *n*-ten Grades lautet:

$$p(x) = \frac{a_0}{2} + \sum_{j=1}^{n} \left(a_j \cdot \cos(j\omega x) + b_j \cdot \sin(j\omega x) \right)$$

7.3 Logarithmische Interpolation

Wenn den zu interpolierenden Punkten eine logarithmische Funktion zugrunde liegt, eignet sich das Verfahren der logarithmischen Interpolation, bei dem die einzelnen Punkte durch logarithmische Funktionen miteinander verbunden werden.[7]

Die mathematische Formulierung lautet:

$$f(x) = f_0 \cdot \exp\left\{ \frac{(x - x_0)(\ln f_1 - \ln f_0)}{x_1 - x_0} \right\}$$

[6] http://www.gaussianprocess.org/

[7] http://www.cmu.edu/biolphys/deserno/pdf/log_interpol.pdf

Quellenverzeichnis

Der Abruf aller genannten Webseiten erfolgte im September und Oktober 2014.

Neben diversen Wikipedia[8] Seiten wurde aus den folgenden Quellen ein allgemeiner Themenüberblick gewonnen:

- http://nm.mathforcollege.com/topics/spline_method.html
- http://www.krucker.ch/skripten-uebungen/IAMSkript/IAMKap3.pdf
- http://www3.math.tu-berlin.de/Vorlesungen/SS09/EinfNumMath/numerik1.pdf
- http://www.youtube.com/watch?v=LFFPbBe7aAs
- http://arxiv.org/pdf/1211.1768.pdf
- http://www.geos.ed.ac.uk/~yliu23/docs/lect_spline.pdf
- http://www.cambridgeincolour.com/tutorials/image-interpolation.htm
- Lambacher Schweizer: Mathematik für berufliche Gymnasien (Jahrgangsstufenband). Ernst Klett Verlag GmbH, Stuttgart, 2008, S. 130f.

Quellen zu spezielleren Informationen wurden im Text als Fußnote angegeben.

Abbildungsnachweis

Alle Abbildungen sind eigene Werke.

Die Graphen wurden mit dem Programm Grapher von Apple, Inc. (Version 2.5) erstellt. Die Gleichungen, Formeln, etc. mit der Software MathType von Design Science, Inc. (Version 6.7e).

Onlineverfügbarkeit

Eine PDF-Version dieses Dokuments steht auf http://www.simsso.de/?weiterleitung=Interpolation zum Download bereit.

[8] http://www.wikipedia.org/